IMDL
EEL5666C
Enforcer
Eduardo Moreno

Instructors:
Dr. A. Antonio Arroyo
Dr. Eric M. Schwartz
TAs : Tim Martin
Josh Weaver

Table of Contents

Abstract

The Enforcer searches for a target, locks on to it and chases it. Its main purpose is obstacle avoidance using sonar sensors. It performs color tracking based on a selected color upon initial calibration. When it approaches the target, it will enforce the law by using its rear piston to inflict a painful strike.

Executive Summary

Enforcer is implemented on a Mavric-II board with an ATmega128 microcontroller. Its body is made out of three sheets of tempered fiberglass held together by two hinges for easy opening of the body. It performs obstacle avoidance by using two forward mounted Ultrasonic Range Finder - Maxbotix HRLV-EZ4 mounted 6 inches apart and pointing away from each other. The reading from these sonars is then averaged in order to eliminate false readings. The CMUcam1 is the second sensors and it allows for color tracking. This is a CMOS camera that is interfaced serially using the USART. Writing commands tells the camera what colors to track and it returns a bounding box containing the limits of the targeted color in its vision window. From there, the microcontroller can suggest which direction to take to approach the target. It moves around using four DC motors powered by a Texas Instrument DRV8833 Dual Motor Driver Carrier using three PWM signals. It calculates the speed of each wheel by keeping track of the last four suggested actions suggested by the sensors. Upon reaching the target it strikes it by using a 1.5" bore single acting pneumatic piston. This piston is powered by a 800 psi CO2 tank regulated down to 120 psi using the AG1 Gen2 Air paintball gun regulator. This is controlled by a solenoid valve which requires 24 volts 1 amp to trigger. The power comes from a switching regulator that can be adjusted from 5-24 volts and controlled by the ATmega128 using a MOSFET transistor IRF510. These parts allow for the Enforcer to perform its task of tracking down an object and striking it with a painful blow.

Introduction

Basic navigational skills, that humans take for granted, are a challenge in robotics. Being able to identify an object form its surroundings to interact with it is part of this problem. Enforcer will be able to do focus color tracking based on its initial calibration. It will chase down the selected color, while avoiding obstacles along the way. Upon reaching the target it will attempt to interact with it by striking it with a pneumatic piston. This interaction can be related to the kicking of a ball or the striking of a punching bag. In this paper we will discuss how this problem was tackled by first talking about the logic and integrated system. Followed by the system platform and actuation, these allow the robot move around in the world. Then the sensors that provide the input for processing and deciding what action to perform based on its surroundings. Finally I will discuss the resulting behavior and experimental observations on the performance of Enforcer.

Integrated System

Enforcer will perform obstacle avoidance via the sonar. This is the primary behavior and overshadows all other behaviors to prevent from hitting or damaging other objects. If it determines that the path is clear it will perform color tracking using the CMUcam. The color is calibrated upon initialization. It will align itself to approach the tracked color as long as there are no obstacles within its sonar sensors. Once it finds an object it will approach it, turn around and strike it with a pneumatic piston located in its rear. The figure #1 shows the block logic diagram.

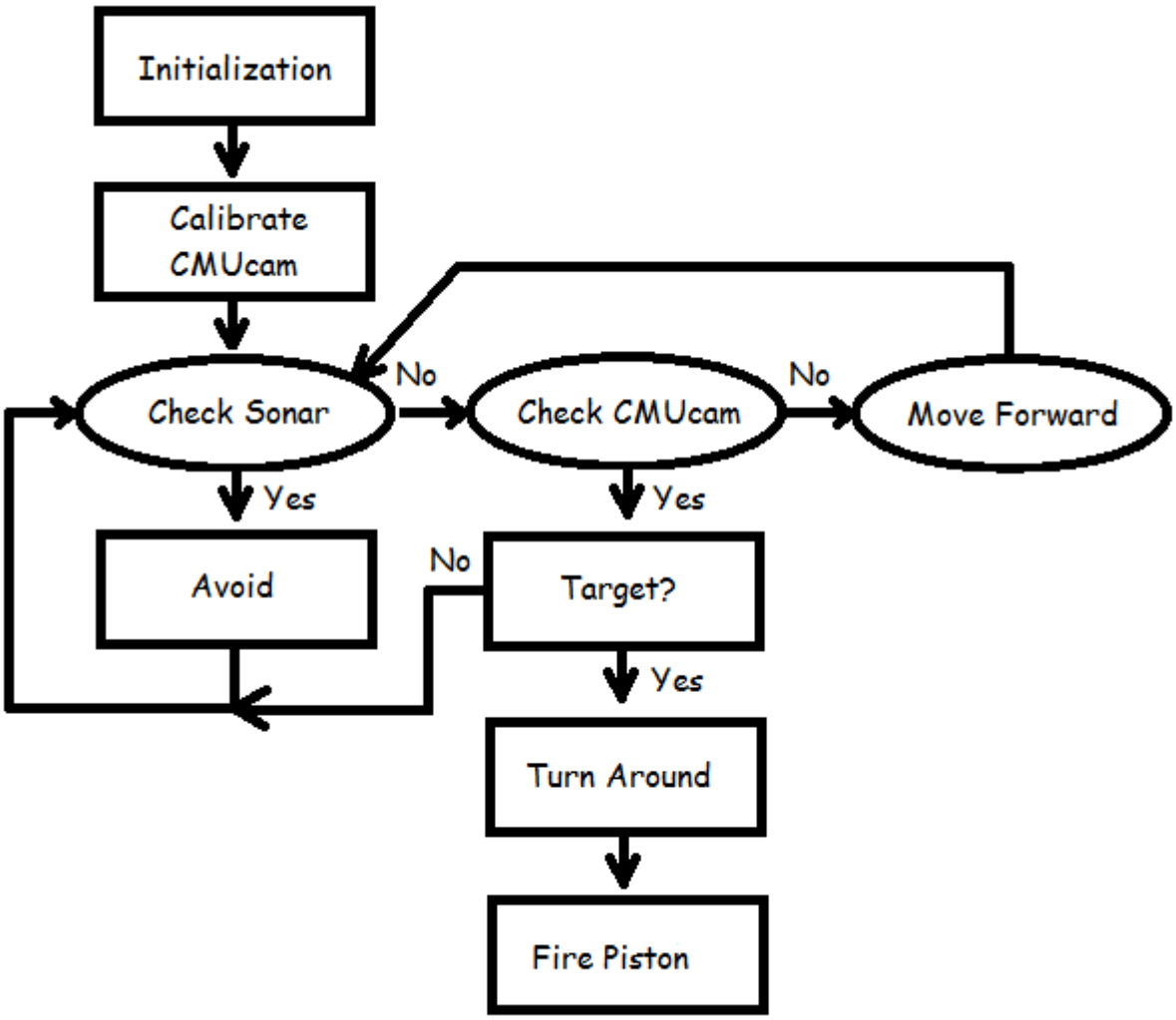

Figure #1

Enforcer is implemented on a Mavric-II board with an ATmega128 microcontroller. I had already purchased this board and it includes everything I need: 3 PWM channels 2 for my wheels and 1 for the servos. Two separate power supplies, a regulator for the microcontroller and a second Vcc to power motors, 8 analog to digital channels for the sonar sensors, serial interface for the CMUcam, two 8 bit timers and one 16 bit timer. The board is small at 2.2x3.6 inches and it contains breakout pins for all the ports.

Mobile Platform

The main platform of the Enforcer is made out of tempered fiberglass ¼ of an inch thick. Three cuts were needed to make the body be able to open up for easy access to all the electronics. Two of the blocks are 12” by 10” the bottom one is the support for all the circuit boards, the motors and the battery. The top one has the LCD and the system for the pneumatic piston. A third piece 2.5” by 7” is the front side of Enforcer and it holds both sonar sensors and the camera pointing forward. All three piece of fiberglass are held together using ¼ inch

diameter screws and 1 inch wide aluminum hinges. A picture of the opened up body fully populated is shown below.

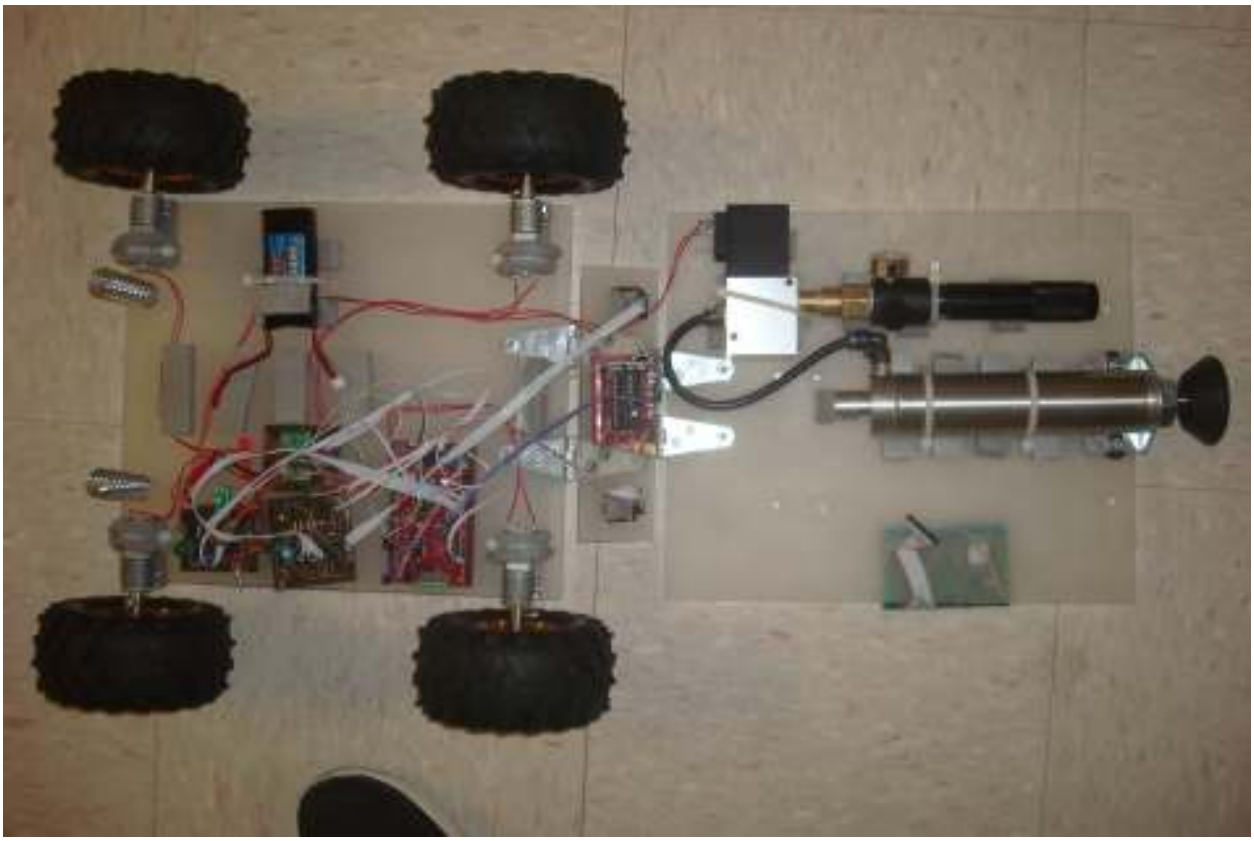

Image #2

Actuation

Enforcer's actuation system is composed of 4 DC motors using thumper wheels. The Wild Thumper wheel provided a 6 inch diameter while have inflatable rubber that helps absorb some of the impact while moving around. The movement will be on four wheels powered by DC motors. These motors are small gearbox motors with 6V and 90 RPM, they supply a 8.6kg-cm torque, 3.3A stall current and 4mm shaft. These motors were chosen due to already having the 4 mm shaft required, their small size (1.72"x0.79"x0.79") and weight and the fact that it contains a metal shaft the same size as the Thumper wheels.

In order to drive the gearbox motors, I purchased two DRV8833 dual H-bridge motor drivers. These motor drivers take a DC voltage from 2.7 to 10.8V and an input PWM signal and can be used to power two DC motors with bidirectional control supplying 1.2 amps to each motor. I chose this motor driver since it operated in the same voltage range as my battery, it has PWM signal controls and additional circuitry to protect the motors from overcurrent and under voltage.

A CO2 Piston Actuator provides the striking mechanism for the Enforcer. It has a bore size of 1.5 inches and a shaft length of 3 inches. It is a single acting piston operating in the open fail state. This means that as long as there is no pressure in the gas chamber, the piston will remain retreated due to an internal spring. A rubber rod foot is attached at the end of the shaft in order to push against a surface. The force of the piston is a push ratio factor of 1.77*psi and operates at 120 psi. Once the pressure is applies then it applies the force to the shaft. The pressure in the piston is generated by using 12 gram CO2 cartridges common used for air soft and paintball guns. It is then regulated by a paintball gun pressure

regulator that brings the pressure down from 50psi to 1500 psi depending on the turn knob. The pressure release is then controlled by a solenoid that triggers at 18 volts and 1 amp. In order to operate this solenoid, a switching regulator had to be built and hooked up using a MOSFET switching transistor IRF510.

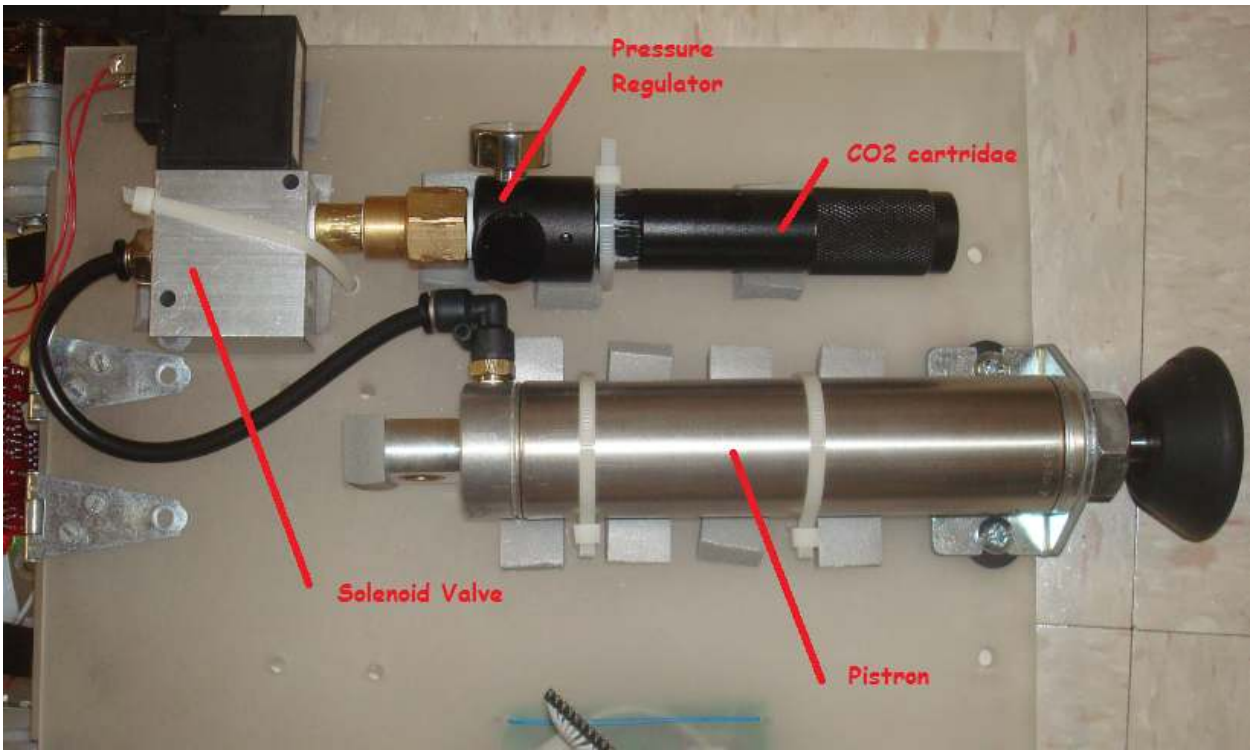

Image #3

The entire system obtains its power form a 7.4 dual cell lithium polymer battery. This battery weights 89 grams, has a power rating of 1A/h and a discharge rate of 25 amps. In order to control the voltage of the battery and prevent permanent damage I made a voltage monitoring circuit. Whenever the battery voltage drops below 7.4 volts an LED turns on. In this same circuit board I also have the MOSFET and the switching regulator circuit I purchased from pololu.

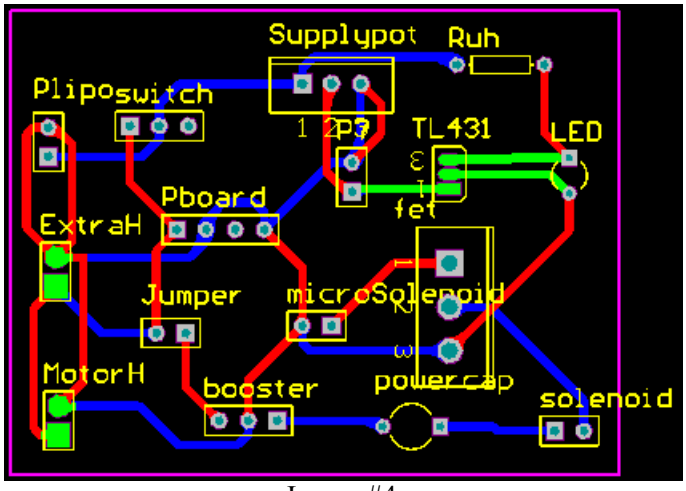

Image #4

Sensors

In order to navigate and perform obstacle detection enforcer will use sonar sensors. This is because the laser range finders were outside my budget at around $200 each. The sensor that was picked out was the Ultrasonic Range Finder HRLV-EZ4. This sonar has an operating voltage of 2.5 to 5.5V and is small and light weight at 4.3 grams. These will be used to determine the range of the object to be jumped on to as well. It features a resolution of 1mm and minimum maximum range of 300 mm and 5000mm respectively. The EZ4 refers to the the smallest possible beam width. Each sonar has to be fired independently in order to reduce the amount of miss reads from nearby units. This is done by firing the first sonar by pulling the Tx pin to ground and then connecting the Rx pin to the following Tx. It is possible to chain multiple units like this in sequence in order reduce interference. In the Enforcer, two HRLV-EZ4 were used in the front of the body. They are fired sequentially in order to obtain the range of objects in the forward direction figure #2 shows a sequential style connection. Even though it is rated for a 1mm resolution experiments showed a 5mm resolution which is enough for the purposes required.

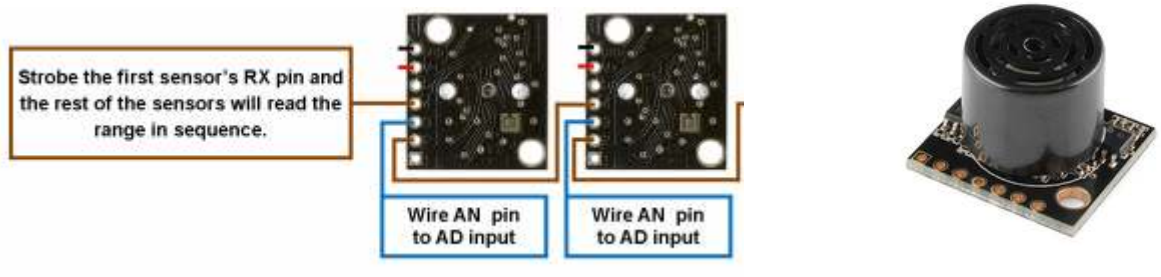

Image #5

A CMUcam1 Vision System is used in the Sand Flea to track a specific color. This color indicates the object or destination to jump on to. I wanted the image processing capability to remain on board. The CMUcam is able to do this. I am using the V1 since it is the one I already had access to in order to cut the total price of the robot. The CMUcam has a java interface GUI that can be used to test the camera. This interface allows you to track the position and size of a colored object. The GUI allows you to program the camera and then you can use it with the Atmega128 using t he USART.

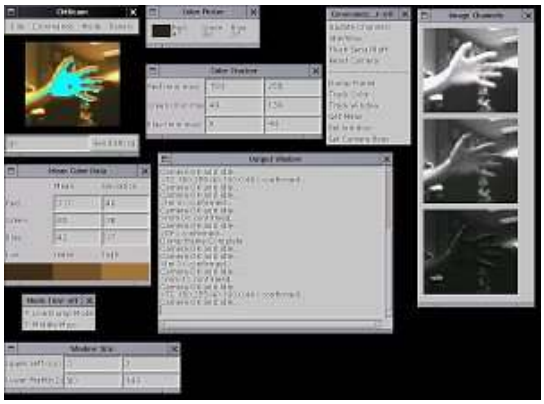

Image #6

The camera communication works by sending a stream of bytes starting with a command, followed by a '\r' character meaning standby mode. Once the camera receives a command it sends back an acknowledgement followed by the corresponding tracking packet. The enforcer has a self calibration mode where upon entering initialization it will ask to present the object to be tracked in front of the camera and take the mean and standard deviation of the RGB 8 bit values of every pixel in the screen and then store it in memory. These values are then used to track the color to be aggressively enforced. The tracking mode starts by the controller sending the byte string "TC Rmin Rmax Gmin Gmax Bmin Bmax\r" where each min value is the color's mean minus the standard deviation (Rmin = Rmean – Rstd) and the max is the mean plus the standard deviation (Rmax = Rmean + Rstd). The camera then returns a packet containing "M mx my x1 y1 x2 y2 pixels and confidence." Mx is obtained and used to guide to robot left or right towards the target, while confidence is used to determine how strong to follow the target and pixel is the amount of pixels in the image within that range in order to determine for sure when standing in front of the object.

Behaviors

Enforcer performs obstacle avoidance by using the sonar to detect objects in its path. The two sonars are pointed slightly away from each other but facing towards the front of the Enforcer. Whenever an object breaks the threshold of being "close" then it reacts by turning left when this is triggered by the right sensor and turning right when triggered by the left sensor. When both of them approach the "tooclose" threshold then it immediately stops the motors and starts backing up. The value of each sensor is the average of the last 4 readings to the sensor. This smoothes out the decision making process and prevents jerky movements.

It also performs color detection with the CMUcam to determine which objects to strike with the piston. Its main purpose is to determine the location in front of it where the object is and correct its pack to head towards it. It should only worry about the object whenever there is nothing within the sonar range. The total image window on the CMUcam is 80x140 pixels. By reading the center of mass of an object returned by Mx we can tell whether its right left or center. A value closer to 0 means head right, 80 means head left and somewhere in the middle would represent a straight path.

The way enforcer knows it found the target to strike is if the center of mass of the object is straight on, is if the Mx value is close to the middle range and the sonar value is triggered into the "tooclose" region. The only way it can approach this region is with a high number of pixels of the same color, otherwise it can be another obstacle located between the robot and the target. Once it finally comes up to the target, it performs a 180 degree turn and fires the piston by triggering the switching regulator to power the solenoid valve. It then waits for the next command to chase another target.

Experimental Layout and Results

In order to calibrate my sonar sensors I mounted them at the height of the robot where they are going to be used and then I used a solid sheet of cardboard to measure the distance being measured along with a meter. I used to function readsonarleft() in the appendix to start a conversion in the left sided sensor. This is done by pulling a start conversion pin high on the sonar for 20 us and then waiting 100ms for the first reading. Once the left sonar is done its Tx pin will go high and it is tied to the start conversion pin of the right sonar which will proceed to perform a single range reading. I was reading the value straight from conversion register and outputting it to the LCD. I placed the cardboard 0.3m from the sonars and recorded the value to be at 59. I did this repeatedly moving the object back 0.1m ever time and noticed that the bit value kept increasing by 20 each time. This means every bit value represents a

resolution of 5mm. This linearity was constant until a value of 360 from the conversion register which was 1.8 meters away. After that the distance was a little more unpredictable and it seemed to increase in an exponential manner. As long as the reading is kept between 0.3 and 1.8 meters, then the sonar can be expected to be precise up to 5mm. The "close" value determined experimentally ended up being 120. This is the distance where the robot still has enough time to correct its course before running into the obstacle. The "tooclose" value is 70, this is the value very close to the dead zone of the sonar when the motors need to be stopped suddenly in order not to hit an obstacle.

Upon trying to control the DC motors I ran into the problem of the movement being very twitchy. My solution was found to be averaging the last motor commands to smoothen out the overall acceleration and movement of the robot. After implementing this I found out the robot weights too much for certain voltage values to break the static friction so I had to put a minimum threshold for the robot to move.

Since the ATMega128 only has 3 output compares and each set of wheels requires at least one I connected them and planned on using them with differential PWMs. The wiring diagram with the chip and the corresponding motors are shown in Image #3. The reason I had to wire it this way is because I needed the full 50% duty cycle PWM power in order to turn in the left and right direction. Instead when going forwards and backwards I only needed half the voltage in order to move since all 4 wheels were working in the same direction. This allowed me to control turning right by turning up PWM signal C and turn left by turning on B and A. Moving forwards required B to be at double the speed of C and backwards required A at double the duty cycle percentage of C.

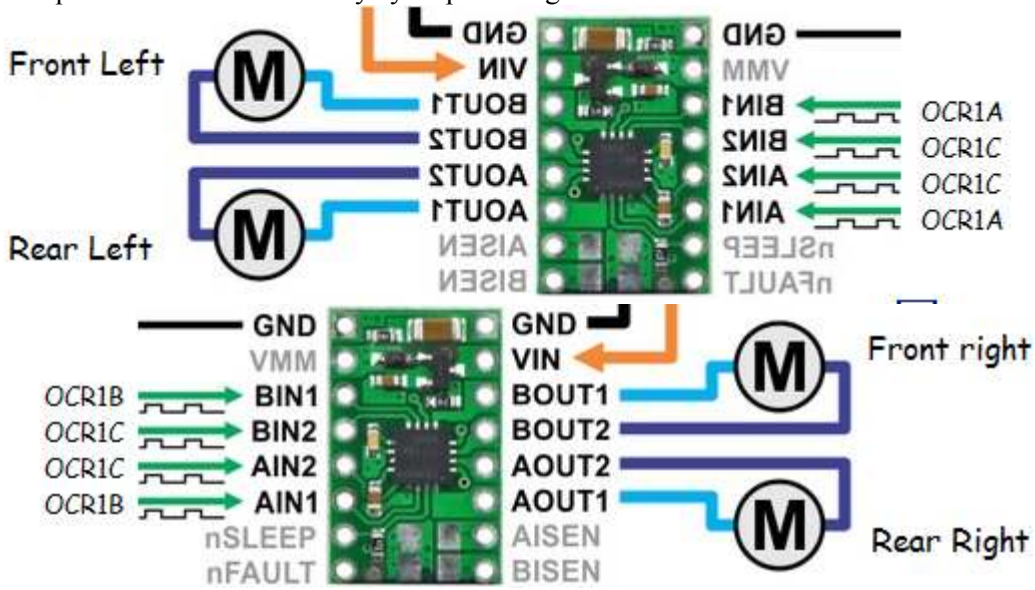

Image #7

For the CMUcam I found that the center of mass is not enough of an indicator to determine the location of the object. I also used the number of pixels found with the same value as the second qualifier that way if the object is near it will be occupying most of the image window if its far it will only be 10 or 20 pixels in size on the 80 by 140 pixel window. I determined experimentally by running the code that a good window for the object being located on the right is from 0 to 25 pixels on the center of mass, left from 55 to 80 and between 25 and 55 its pretty much straight on or "locked in."

Conclusion

Given out the start of the semester, I think I set my goals a little too high by attempting to make the Sand Flea. This gave to a lot of frustration and disappointment leading to the day before pre-demo, when I set myself back to almost zero. For the small amount of time since then, in 2 weeks I am satisfied

with the progress on the Enforcer. As a final robot however I feel it is a basic robot and I wish I had spend more time on it to make it do other complex functions. However I still have my sight set on the Sand Flea and would like to someday modify it to perform such tasks.

I am proud of the wheel encoding system I came up with on such a short notice when I noticed my robot wasn't able to turn due to not having enough power on the DC motors. This allows me to turn with straight PMWs and use differentials to create the forward and backwards motion. Another area I am proud of is all the code is written by me from scratch, I decided that in order to determine how your robot works it's better to generate code yourself than to use libraries. The downfall of this is that the code needs improvement and some cleaning up in order to make it user friendly, as it stands right now it is a bunch of functions calling each other in one main file. An area that would need to be improved on the enforcer is making the air tank on the piston last longer, that way it is easier to get more runs in without having to change the cartridge.

My future work requires some IR sensors to make the robot have object sensing while backing up and align the piston better with the target. This is however a small task that can be completed in a short amount of time. A larger goal would be to improve the piston system, perhaps change the source of air from CO2 cartridges into something else with less pressure or an electrical piston. The longer term goals include upgrading Enforcer to be able to jump with the piston. This includes a more sturdy body, better shock absorption and a way to handle impact on the wheel motors. The tilt sensors are already implemented inside the circuit boards and can be hooked up as well as the code for the tilt sensors and the servos is already completed, only the servos need to be mounted to the robot.

Documentation

[1] Sonar - http://www.maxbotix.com/performance.htm

[2] CMUcam - http://www.cmucam.org/projects/cmucam1

[3] Piston - http://www.frightprops.com/pneumatics/cylinders/least-common/single-acting/single-acting-universal-mount/1-1-2-bore-single-acting-universal-mount-0986-1500.html

[4] Atmel128 full manual - http://www.atmel.com/Images/doc2467.pdf

[5] IMDL class handouts - http://mil.ufl.edu/5666/handouts.htm

Appendix

```
/* Created: 11/26/2012 10:56:45 AM
 *  Author: EM
 */
#define F_CPU 16000000L
//#include <asf.h>
#include <avr/io.h>
#include <stdlib.h>
#include <util/delay.h>
#include <string.h>
#include "LCD.h"

#define BAUD 19200 //jumper3 //38400 jumper 2 //115200 no funciona
#define MYUBRR F_CPU/16/BAUD-1
uint8_t cmudata[20];

void pmw_init(void);
```

```
void atod_init(void);
void motorcontrol(uint16_t,uint16_t,uint16_t);
void backup(uint16_t);
void forward(uint16_t);
void right(uint16_t);
void left(uint16_t);
void stop(void);
uint8_t avoid(uint8_t);
void turnaround(void);
//void servoangle(int8_t);
uint16_t readsonarright(void);
uint16_t readsonarleft(void);
void fireinthehole(void);
//int tilting(void);
void cmucam_init(unsigned int);
void USARTwrite(char *data);
void USARTread(char);
void TrackColor(void);
void calibrateCAM(void);

uint16_t motorleft[4];
uint16_t motorright[4];
uint16_t motorback[4];
uint16_t sonarleft[4];
uint16_t sonarright[4];
uint16_t sonarr,sonarl;
uint8_t red,green,blue,stdr,stdg,stdb;
uint8_t mx, my, x1, y1, x2, y2, pixels, confidence;
uint16_t straightleft=1000;
uint16_t straightright=1000;
uint8_t close=120;
uint8_t tooclose=70;
uint8_t speed=5; //0 stop 10 max speed
uint16_t i,n,object;

int main(void){
//uint32_t,int16_t

	WDTCR |= (1<<WDCE) | (1<<WDE);//turn watchdog timer off
	WDTCR=0x00;
	DDRA=0x01; //solenoid
	PORTA=0x00;

	LCD_init();
	LCD("Initializing",1);

	pmw_init();
	atod_init();
	//LCDshow(MYUBRR,2);
	cmucam_init(MYUBRR);
	calibrateCAM();
	LCD_comm(0x01);
	LCD("Calibration Successful",1);
	_delay_ms(2000);
	LCD("Eduardo Moreno",1);
	LCD("Enforcer",2);

	while(bit_is_set(PINC,7)){_delay_us(1);}
```

```
//            for(i=0;i<4;i++){
//                              readsonarleft();
//                              readsonarright();
//            }
while(1){

              TrackColor();
              //LCD_comm(0x01);
              sonarl=readsonarleft();
              sonarr=readsonarright();
//            LCDshow(sonarl,1);
//            LCDshow(sonarr,2);
//            _delay_ms(3000);

              if( mx!=0 && my!=0 && confidence>50){
                            LCD("found target",1);
                     if(mx>55 && mx<80){
                            object=avoid(0);
                            if(object==1){
                                   LCD("evading",1);
                            }
                            else{
                                   LCD("target on left",2);
                                   left(speed);
                            }
                     }
                     else if(mx>0 && mx<25){
                            object=avoid(0);
                            if(object==1){
                                   LCD("evading",1);
                            }
                            else{
                                   LCD("target on right",2);
                                   right(speed);
                            }
                     }
                     else if(mx>=25 && mx<=55){
                            if(sonarl<close && sonarr<close){
//                                 while(sonarl>tooclose && sonarr>tooclose){
//                                        readsonarleft();
//                                        readsonarright();
//                                        forward(speed);
//                                        forward(0);
//                                 }
                                   turnaround();
                                   _delay_ms(1000);
                                   LCD("fire!",2);
                                   fireinthehole();
                                   while(bit_is_set(PINC,7)){_delay_us(1);}
                            }
                            else{
                                   LCD("locked in",2);
                                   forward(speed);
                            }
                     }
              }
              else{
                            LCD("searching for target",1);
```

```
                        avoid(1);
                }
        }
}

void atod_init(void){
        DDRF=0x04;
        PORTF = 0x00;
        ADMUX=0x01;
        ADMUX|=(1 << REFS0); // Set ADC reference to AVCC
        ADCSRA|=(1 << ADPS1)|(1 << ADPS0);//(1 << ADPS2) //prescalar clk/128=125k
        //ADCSRA|=(1 << ADFR);  // Set ADC to Free-Running Mode
        ADCSRA|=(1 << ADEN);  // Enable ADC
        _delay_us(25);//needs 13 clocks after AD is enabled
        for(i=0;i<4;i++){
                sonarright[i]=100;
                sonarleft[i]=100;
        }
}

void pmw_init(void){
        TCCR1A = 0xA8; // OCA and OCB toggle on match
        TCCR1B = 0x12; // PWM correct phase //clock/8 prescaler
        DDRB = 0xE0; //outputs
        ICR1 = 20000; //max value for timer counter
        TCNT1 = 0x0000; //clear 16 bit counter
        OCR1A = 0; //initial values 0 pin 5
        OCR1B = 0; //pin 6
        OCR1C = 0; //pin 7
        for(i=0;i<4;i++){
        motorright[i]=0;
        motorleft[i]=0;
        motorback[i]=0;
        }
}

void motorcontrol(uint16_t left, uint16_t right, uint16_t back){
        uint16_t sumright=0;
        uint16_t sumleft=0;
        uint16_t sumback=0;
        for(i=3;i>0;i--){
                motorleft[i]=motorleft[i-1];
                motorright[i]=motorright[i-1];
                motorback[i]=motorback[i-1];
        }
        motorleft[0]=left;
        motorright[0]=right;
        motorback[0]=back;
        for(i=0;i<4;i++){
                sumleft+=motorleft[i];
                sumright+=motorright[i];
                sumback+=motorback[i];
        }
        //LCDshow(sumright,2);
        if(sumback<5001){
                OCR1A=0;}
        else{
                OCR1A = sumback>>2;}
```

```
	if(sumright<5001){
		OCR1B=0;}
	else{
		OCR1B = sumright>>2;}
	if(sumleft<5001){
		OCR1C=0;}
	else{
		OCR1C = sumleft>>2;}
}

void backup(uint16_t amount){
	motorcontrol(amount*1000, 0, amount*2000);
}
void forward(uint16_t amount){
	//C B A
	// C L1 R1 B
	// A L2 R2 C
	motorcontrol(amount*1000, amount*2000, 0);
}
void left(uint16_t amount){
	motorcontrol(0, amount*2000, amount*2000);
}

void right(uint16_t amount){
	motorcontrol(amount*2000, 0, 0);
}
void stop(void){
	OCR1A= 0;
	OCR1B= 0;
	OCR1C= 0;
	motorright[0]=0;
	motorleft[0]=0;
	motorback[0]=0;
	motorright[1]=0;
	motorleft[1]=0;
	motorback[1]=0;
	motorright[2]=0;
	motorleft[2]=0;
	motorback[2]=0;
}

uint8_t avoid(uint8_t avoiding){
	uint8_t obstacle=1;
if(sonarr<close || sonarl<close){
	if(sonarr<tooclose || sonarl<tooclose){
		stop();
		LCD("obstacle near",2);
		do{
		backup(speed);
		sonarl=readsonarleft();
		sonarr=readsonarright();
		}while(sonarl<tooclose -5  || sonarr<tooclose-5);
		stop();
	}
	if(sonarr<=sonarl){
		LCD("obstacle on right",2);
		left(speed);}
	else if(sonarr>sonarl){
```

```
                LCD("obstacle on left",2);
                right(speed);}
        else if(sonarr<close && sonarl<close){ //first
                LCD("obstacle ahead",2);
                backup(speed);}
}
else{
        LCD("path clear",2);
        if(avoiding==1){forward(speed);}
        else{obstacle=0;}
        }
        return obstacle;
}

void turnaround(void){
        stop();
        LCD_comm(0x01);
        LCD("turning around",1);
        left(speed);
        _delay_ms(5500);
        stop();
}

uint16_t readsonarleft(void){
        char h,l;
        uint16_t sumsleft=0;
        ADMUX |= 0x01;
        PORTF |= 0x04; //bring pin 2 high 20us to start conversion
        _delay_us(20);
        PORTF &= 0xFB;
        _delay_ms(100);//data is available 100 ms later
        ADCSRA|=(1 << ADSC);  // Start AtoD Conversions
{}while(ADCSRA & 0x40);
        l=ADCL;
        h=ADCH;
        sonarleft[0]=(h<<8)+l;
        for(i=3;i>0;i--){
                sonarleft[i]=sonarleft[i-1];
        }
        sonarleft[0]=(h<<8)+l;
        for(i=0;i<4;i++){
                sumsleft+=sonarleft[i];
        }
        //LCDshow(sonarleft[0],1);
        return (sumsleft>>2);
}

uint16_t readsonarright(void){
        char h,l;
        uint16_t sumsright=0;
        ADMUX &= 0xF0;
                PORTF |= 0x08; //bring pin 2 high 20us to start conversion
                _delay_us(20);
                PORTF &= 0xF7;
                _delay_ms(100);//data is available 100 ms later
        ADCSRA|=(1 << ADSC);  // Start AtoD Conversions
        {}while((ADCSRA & 0x40));
        l=ADCL;
```

```
	h=ADCH;
	for(i=3;i>0;i--){
		sonarright[i]=sonarright[i-1];
	}
	sonarright[0]=(h<<8)+l;
	for(i=0;i<4;i++){
		sumsright+=sonarright[i];
	}
	//LCDshow(sonarright[0],1);
	return (sumsright>>2);
}

/*void servoangle(int8_t angle){
	// 1ms pwm = -90 degrees = 1000
	// 1.5ms pwm = 0 degrees = 1500
	// 2ms pwm = 90 degrees = 2000
	OCR1A=1500+((angle/90.0)*500.0);
	//_delay_ms(3000);
	//OCR1A=1500;
}*/

/*int tilting(void){
	DDRD &= 0x9F;
	if(PIND & 0x01){
		LCD("up",1);
		return 1;}
	else{LCD("down",1);
	return 0;}
	//if(PORTD & 0x02){LCD("up",2);}
	//else{LCD("down",2);}
}*/

void cmucam_init(unsigned int ubrr){
	DDRE=0x02; //set pinE0 RXD0 input, pinE1 TXD0 output
	/* Set baud rate */
	UBRR0H = (unsigned char)(ubrr>>8);
	UBRR0L = (unsigned char)ubrr;
	/* Enable receiver and transmitter */
	UCSR0B = (1<<RXEN)|(1<<TXEN);
	/* Set frame format: 8data, 1 stop bit, asynchronous, no parity*/
	UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);

	USARTwrite("RS\r"); // Reset CMUcam
	USARTread(':');
	_delay_ms(5000);
	USARTwrite("L1 1\r"); //light on
	USARTread(':');
	_delay_ms(5);
	USARTwrite("PM 1\r"); // Set CMUcam to Poll Mode
	USARTread(':');
	_delay_ms(5);
	//USARTwrite("NF 1\r"); // turn on noise filtering
	//USARTread(':');
	//_delay_ms(5);
	USARTwrite("L1 2\r"); //light on
	USARTread(':');
	_delay_ms(5);
}
```

```
void USARTwrite(char *data){
int i = 0;
do{
       while (!(UCSR0A & (1<<UDRE0)) ); //empty transmit buffer
       UDR0 = data[i]; // store data in tx_char
       while (!(UCSR0A & (1<<TXC0)) ); //transmit complete
}while(data[i++]!='\r');
}

void USARTread(char ending){
       memset( cmudata, 0, 10);
int i = 0;
do{
       while (!(UCSR0A & (1<<RXC0)) ); //wait for data to be received
       cmudata[i] = UDR0;
  }while(cmudata[i++]!=ending);
}

void TrackColor(void){
char camcom[26];
char num[3];
int n;
int k=4;
uint8_t rmin,rmax,gmin,gmax,bmin,bmax;

if(red-stdr<0){rmin=0;}
else{rmin=red-stdr;}
if(red+stdr>255){rmax=255;}
else{rmax=red+stdr;}
if(green-stdg<0){gmin=0;}
else{gmin=green-stdg;}
if(green+stdg>255){gmax=255;}
else{gmax=green+stdg;}
if(blue-stdb<0){bmin=0;}
else{bmin=blue-stdb;}
if(blue+stdb>255){bmax=255;}
else{bmax=blue+stdb;}
     strcpy(camcom, "TC ");
     itoa(rmin,num,10);
     strcat(camcom, num);
        strcat(camcom, " ");
             itoa(rmax,num,10);
             strcat(camcom, num);
             strcat(camcom, " ");
     itoa(gmin,num,10);
     strcat(camcom, num);
        strcat(camcom, " ");
             itoa(gmax,num,10);
             strcat(camcom, num);
             strcat(camcom, " ");
     itoa(bmin,num,10);
     strcat(camcom, num);
        strcat(camcom, " ");
             itoa(bmax,num,10);
             strcat(camcom, num);
        strcat(camcom,"\r");
do{
```

```
USARTwrite(camcom);
USARTread(':');
//TC [Rmin Rmax Gmin Gmax Bmin Bmax]\r
//M mx my x1 y1 x2 y2 pixels confidence\r
//C x1 y1 x2 y2 pixels confidence\r
	if(cmudata[k+3]==' '){ //mx
		mx=(0x0F&cmudata[5]);
		n=k;
	}
	else {
		if(cmudata[k+4]==' '){n=k+1;}
		else{n=k+2;}
		mx=(0x0F&cmudata[n])*100+(0x0F&cmudata[n+1])*10+(0x0F&cmudata[n+2]);
	}
	if(cmudata[n+5]==' '){ //my
		my=(0x0F&cmudata[n+4]);
		n=n+2;
	}
	else {
		if(cmudata[n+6]==' '){n=n+3;}
		else{n=n+4;}
		my=(0x0F&cmudata[n])*100+(0x0F&cmudata[n+1])*10+(0x0F&cmudata[n+2]);
	}
	if(cmudata[n+5]==' '){ //x1
		x1=(0x0F&cmudata[n+4]);
		n=n+2;
	}
	else {
		if(cmudata[n+6]==' '){n=n+3;}
		else{n=n+4;}
		x1=(0x0F&cmudata[n])*100+(0x0F&cmudata[n+1])*10+(0x0F&cmudata[n+2]);
	}
	if(cmudata[n+5]==' '){ //y1
	y1=(0x0F&cmudata[n+4]);
	n=n+2;
	}
	else {
		if(cmudata[n+6]==' '){n=n+3;}
		else{n=n+4;}
		y1=(0x0F&cmudata[n])*100+(0x0F&cmudata[n+1])*10+(0x0F&cmudata[n+2]);
	}
	if(cmudata[n+5]==' '){ //x2
		x2=(0x0F&cmudata[n+4]);
		n=n+2;
	}
	else {
		if(cmudata[n+6]==' '){n=n+3;}
		else{n=n+4;}
		x2=(0x0F&cmudata[n])*100+(0x0F&cmudata[n+1])*10+(0x0F&cmudata[n+2]);
	}
	if(cmudata[n+5]==' '){ //y2
		y2=(0x0F&cmudata[n+4]);
		n=n+2;
	}
	else {
		if(cmudata[n+6]==' '){n=n+3;}
		else{n=n+4;}
		y2=(0x0F&cmudata[n])*100+(0x0F&cmudata[n+1])*10+(0x0F&cmudata[n+2]);
```

```
        }
        if(cmudata[n+5]==' '){ //pixels
        pixels=(0x0F&cmudata[n+4]);
        n=n+2;
        }
        else {
        if(cmudata[n+6]==' '){n=n+3;}
               else{n=n+4;}
               pixels=(0x0F&cmudata[n])*100+(0x0F&cmudata[n+1])*10+(0x0F&cmudata[n+2]);
        }
        if(cmudata[n+5]==' '){ //caonf
        confidence=(0x0F&cmudata[n+4]);
        n=n+2;
        }
        else {
        if(cmudata[n+6]==' '){n=n+3;}
               else{n=n+4;}

        confidence=(0x0F&cmudata[n])*100+(0x0F&cmudata[n+1])*10+(0x0F&cmudata[n+2]);
        }
        }while(cmudata[k]!='M');
//  LCD_comm(0x01);
//  LCD(cmudata,1);
//  LCDshow(mx,2);

// while(bit_is_set(PINC,7)){_delay_us(1);}
}

void fireinthehole(void){
        PORTA |= 0x01;
        _delay_ms(200); //500 sounds good
        PORTA &= 0x00;
}

void calibrateCAM(void){
        int n;
        int k=4;
        //S rMean, gMean, bMean, rDev, gDev, bDev\r
        LCD("Calibrate Camera",1);
        LCD("press button",2);
        while(bit_is_set(PINC,7)){_delay_us(1);}
        do{
        USARTwrite("GM\r"); //get mean color
        USARTread(':');
        if(cmudata[k+3]==' '){ //red
               red=(0x0F&cmudata[5]);
               n=k;
        }
        else {
               if(cmudata[k+4]==' '){n=k+1;}
               else{n=k+2;}
               red=(0x0F&cmudata[n])*100+(0x0F&cmudata[n+1])*10+(0x0F&cmudata[n+2]);
        }
        if(cmudata[n+5]==' '){ //green
               green=(0x0F&cmudata[n+4]);
               n=n+2;
        }
        else {
```

```
                if(cmudata[n+6]==' '){n=n+3;}
                else{n=n+4;}
                green=(0x0F&cmudata[n])*100+(0x0F&cmudata[n+1])*10+(0x0F&cmudata[n+2]);
        }
        if(cmudata[n+5]==' '){ //blue
                blue=(0x0F&cmudata[n+4]);
                n=n+2;
        }
        else {
                if(cmudata[n+6]==' '){n=n+3;}
                else{n=n+4;}
                blue=(0x0F&cmudata[n])*100+(0x0F&cmudata[n+1])*10+(0x0F&cmudata[n+2]);
        }
        if(cmudata[n+5]==' '){ //stdr
        stdr=(0x0F&cmudata[n+4]);
        n=n+2;
        }
        else {
                if(cmudata[n+6]==' '){n=n+3;}
                else{n=n+4;}
                stdr=(0x0F&cmudata[n])*100+(0x0F&cmudata[n+1])*10+(0x0F&cmudata[n+2]);
        }
        if(cmudata[n+5]==' '){ //stdg
                stdg=(0x0F&cmudata[n+4]);
                n=n+2;
        }
        else {
                if(cmudata[n+6]==' '){n=n+3;}
                else{n=n+4;}
                stdg=(0x0F&cmudata[n])*100+(0x0F&cmudata[n+1])*10+(0x0F&cmudata[n+2]);
        }
        if(cmudata[n+5]==' '){ //stdb
                stdb=(0x0F&cmudata[n+4]);
                n=n+2;
        }
        else {
                if(cmudata[n+6]==' '){n=n+3;}
                else{n=n+4;}
                stdb=(0x0F&cmudata[n])*100+(0x0F&cmudata[n+1])*10+(0x0F&cmudata[n+2]);
        }
        }while(cmudata[k]!='S');
}
```